САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ – ПРОЦЕССОВ УПРАВЛЕНИЯ

КАФЕДРА МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ ЭНЕРГЕТИЧЕСКИХ СИСТЕМ

Купинская Ася Игоревна

Выпускная квалификационная работа бакалавра

Задача маршрутизации вывоза и доставки товара с несколькими транспортными средствами

Направление 01.03.02

Прикладная математика, фундаментальная информатика и программирование

Научный руководитель,

кандидат физ.-мат. наук,

доцент

Власова Т.В.

Санкт-Петербург

2019

Содержание

Введение

Комбинаторная оптимизация занимается поиском оптимального объекта в конечном множестве объектов. Одной из известных задач этой области является задача коммивояжера (Travelling salesman problem, TSP). Коммивояжер (от фр.*commis voyageur)* – разъездной посредник, торговый агент фирмы, передвигающийся с поручениями. Его задача объехать все заданные пункты. Необходимо составить наиболее выгодный маршрут, обходящий все точки и возвращающийся в исходную.

Основной целью в транспортной логистике является построение эффективных с точки зрения стоимости маршрутов объезда транспортными средствами пунктов-продавцов и пунктов-покупателей. В понятие стоимости в данном случае включаются любые затраты, возникающие в процессе объезда клиентов транспортным средством (ТС), которые могут определяться через длину маршрута, время в пути или количество потребляемого топлива. Правильная организация транспортировки может уменьшить экономические затраты на перевозку. Поэтому этот вопрос является одним из наиболее важных для многих предприятий.

Все задачи данного класса являются NP-трудными, поэтому в тестовых примерах большой размерности задача становится трудно вычисляемой, т.е. не может быть решена методами с полным перебором вариантов решений. Как правило, используют эвристические и метаэвристические алгоритмы. Они генерируют решения, приближенные к оптимальному, но за меньшее по сравнению с точными методами время.

В данной работе рассмотрен один из видов обобщения задачи коммивояжера – задача маршрутизации с вывозом и доставкой несколькими транспортными средствами. В главе 1 представлена классификация задач из данного класса, а в главе 2 – методы решения, и подробно разобраны метод

поиска с запретами и генетический алгоритм для построения маршрутов. Глава 3 посвящена одному из критериев оценки эффективности эвристического алгоритма – временной состоятельности решений и методу модификации алгоритма, идея которого основана на временной несостоятельности решений, получаемых эристиками. В главе 4 приведена формулировка задачи маршрутизации 1-1 с вывозом и доставкой с несколькими транспортными средствами одинаковой грузоподъемности, а затем представлены решения задачи на тестовых данных и анализ полученных решений.

Целью работы является решение задачи 1-1 маршрутизации вывоза и доставки товара с несколькими транспортными средствами одинаковой грузоподъемности.

Задачи работы:

- Исследование решений задачи транспортной маршрутизации, генерируемых эвристическими методами;
- Программная реализация метода поиска с запретами и генетического алгоритма;
- Разработка динамической адаптации метода поиска с запретами;
- Сравнение результатов работы алгоритмов между собой.

Глава 1. Обзор задач транспортной маршрутизации с вывозом и доставкой товара (Vehicle Routing Problem with Pickups and Deliveries)

Задача транспортной маршрутизации (Vehicle Routing Problems, VRP) – это большой раздел логистики, в которой необходимо перевезти груз (товар, пассажиров и т. д.) из пунктов вывоза в пункты назначения. В 1959 году Г. Данциг и Дж. Рамсер [1] ввели обобщение задачи коммивояжера: рассмотрели задачу отправки груза (The Truck Despatching Problem). В этой задаче необходимо найти минимальный маршрут для грузового автомобиля, начинающийся в заданной точке, проходящий через все заданные пункты один раз и возвращающейся в исходный пункт в конце пути. При этом было введено дополнительное условие – ограничение грузоподъемности автомобиля. Задача имеет большую практическую значимость, поэтому с тех пор было проведено множество исследований, посвященных различным вариантам данной проблемы.

В данной работе рассматривается класс задач транспортной маршрутизации с вывозом и доставкой товара (Vehicle Routing Problem with Pickups and Deliveries, VRPPD) [2]. Особенностью этого класса является наличие двух типов клиентов – поставщиков и потребителей. Поставщики (pickups) – клиенты, производящие товар, который у них необходимо забрать. Потребители (deliveries) – клиенты, которым необходимо доставить товар, полученный от поставщиков. Задачи, относящиеся к этому разделу, можно разделить на три группы по взаимосвязям между клиентами [2,3]: от многих ко многим (M-M) [4,5], от одного ко многим к одному (1-M-1) [6], от одного к одному (1-1) [7,8].

1.1. Задачи транспортной маршрутизации группы M-M с вывозом и доставкой товара (Many-to-many Vehicle Routing Problem with Pickups and Deliveries)

В такой задаче любой пункт может быть пунктом назначения и пунктом отправки одновременно. Часто на практике требуется в одной остановке как выгрузить груз, так и загрузить новый. Именно такие ситуации моделируются в данной группе. Примером будет являться задача обмена [9]. В этой задаче каждый заказчик определяет количество товара, которое он хочет получить и продать. Цель задачи: построить такой маршрут, чтобы после его выполнения во все вершины было доставлено необходимое количество товаров.

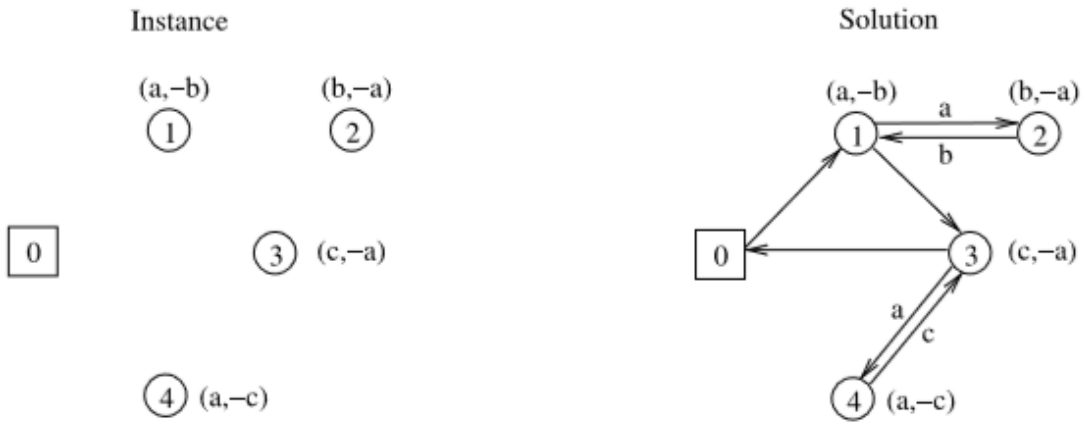

Рисунок 1.1. Пример задачи типа M-M с вершинами {0,1,2,3,4} и товарами {a, b, c}. Метки вершин (x, −y) означают, что вершина предоставляет x и требует y. Метки дуги обозначают товар, перевозимый ТС[3].

1.2. Задачи транспортной маршрутизации группы 1-M-1 с вывозом и доставкой товара (One-to-many-to-one Vehicle Routing Problem with Pickups and Deliveries)

В этой задаче часть товаров (deliveries) направляется со склада в вершины клиентов, а другая часть (pickups), поставляемая клиентами, возвращается в депо. Например, задачи обратной логистики: когда полные контейнеры необходимо доставить покупателям, а пустые должны возвращаться на склад. И. Грибковская и Г. Лапорт описывают применение такой модели при

поставках морских нефтегазовых платформ [10].

Задачи данного класса могут иметь различные структуры спроса: комбинированные требования (хотя бы один клиент требует доставить ему товар и забрать от него другой) и отдельные требования (в этом случае все клиенты разделены по требованиям). Кроме того можно накладывать ограничения на порядок обхода поставщиков и потребителей. Например, запросы deliveries должны быть обслужены после того, как обслужены все запросы pickups [3].

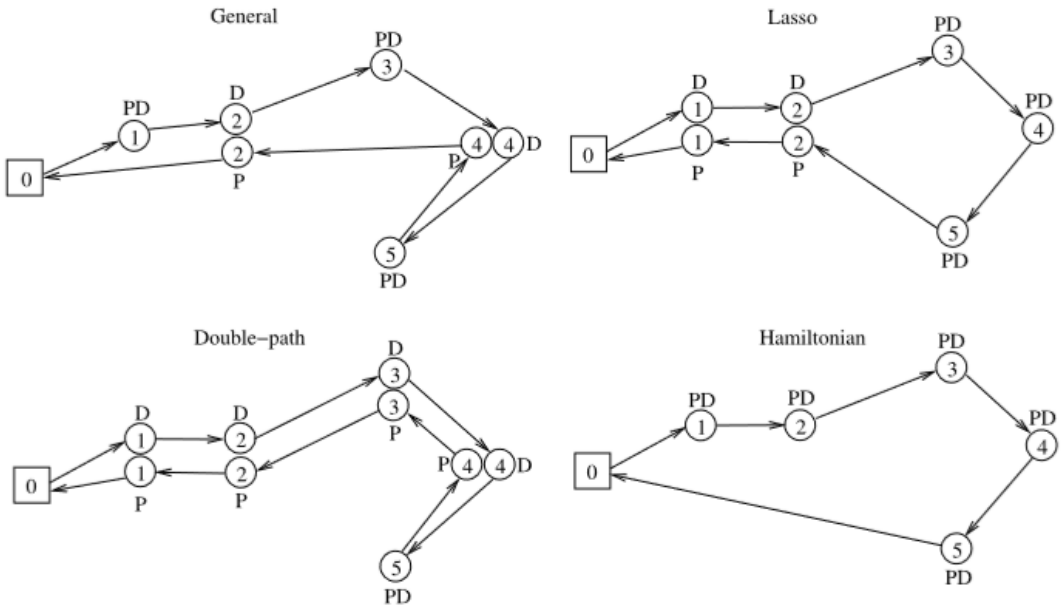

Рисунок 1.2. Пример форм решений задачи 1-M-1, P обозначает вывоз товара из пункта, D – доставку и PD – одновременная доставка и вывоз[3].

1.3. Задачи транспортной маршрутизации группы 1-1 с вывозом и доставкой товара (One-to-one Vehicle Routing Problem with Pickups and Deliveries)

В условиях этой задачи для каждого товара есть вершина-продавец и соответствующая ей вершина-потребитель. Эти проблемы являются n-товарными, так как существует взаимно-однозначное соответствие между вершинами получения и доставки. Необходимо произвести доставку из пункта отправки в пункт назначения. Примером такой задачи является курьерская доставка. Курьеру необходимо обслужить пары поставщик-

потребитель. К данному классу относятся два типа задач – задача маршрутизации транспортного средства с вывозом и доставкой (VRPPD) и ее аналог задача адресной перевозки (DARP, Dial-a-ride problem [11]). VRPPD занимается перевозкой объектов, а DARP – пассажиров, например, такси или скорая помощь. В последней задаче учитываются требования пользователей [2]. Еще одна задача типа 1-1 – задача крана-штабелера (SCP, Stacker crane problem). Особенностью является то, что ТС (кран) одновременно может переносить только одно грузовое место [12].

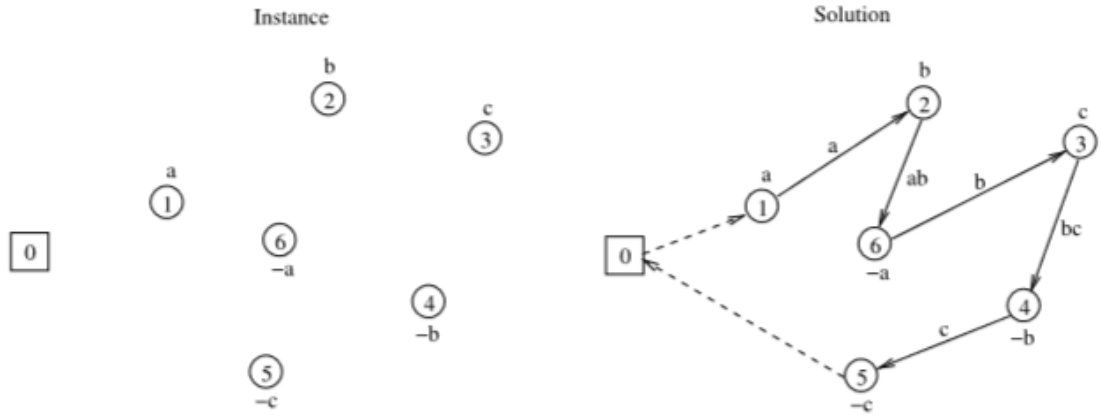

Рисунок 1.3. Пример задачи типа 1-1 [3].

1.4. Виды ограничений в задачах VRP

Задача маршрутизации называется статической [3], когда все данные известны до построения маршрута. А в динамических задачах [2] часть данных поступает, когда движение по маршруту уже началось. В этом случае нельзя рассчитать маршрут для всей задачи, а можно определить стратегию решения, которая на основе полученной информации определяет дальнейшие действия.

Существует множество дополнительных ограничений, накладываемых на условия задачи.

1. Ограничение на количество ТС

   Выделяют задачи с одним ТС [5, 6], несколькими одинаковыми ТС [11] и несколькими различными ТС [7].

2. Ограничение на грузоподъемность ТС (Capacitated VRP) [13]

   В задачах такого типа объем груза для каждого ТС в любой момент времени не должен превышать заданную величину (она может быть одинаковая для всех либо для каждого своя). В таких задачах помимо минимизации затрат на маршруты можно минимизировать количество ТС, необходимое для решения задачи.

3. Ограничение по времени (VRP with Time Windows) [14]

   Для каждого заказчика выделяется временное окно, в которое он должен быть обслужен. И тогда возникают ограничения:

   1) решение является не допустимым, если клиент обслуживается после верхней границы временного окна;
   2) если ТС прибыло в вершину раньше наступления времени начала обслуживания, то необходимо дождаться его наступления.

4. Задачи с несколькими депо (Multiple Depot VRP) [15]

   В данной задаче имеется несколько депо. При построении маршрутов необходимо учесть, что каждое ТС должно начинать и заканчивать свой путь в одном и том же депо.

5. Задачи с разделенным обслуживанием (Split Delivery VRP) [16]

   Снимается ограничение, что один клиент обслуживается только одним ТС. Парк ТС включает машины различной грузоподъемности.

6. Периодическая маршрутизация (Periodic VRP) [17]

   Обобщение классической задачи путем расширения периода планирования до нескольких дней.

7. Задачи со случайными данными (Stochastic VRP) [18]

   Часть компонент задачи (клиенты, запросы, расстояния и т.д.) являются случайными величинами. В данных задачах невозможно требовать выполнения всех ограничений, поэтому для части условий требуется выполнение их с некоторой вероятностью.

Глава 2. Методы решения задачи транспортной маршрутизации с вывозом и доставкой

Задача VRP относится к классу NP-трудных. Это означает, что ее вычислительная сложность зависит от размера входных данных экспоненциально. Для решения используются алгоритмы, которые делятся на три группы: точные, эвристические и метаэвристические.

При небольших размерах задач используют точные методы:

- метод ветвей и границ [16, 19];
- метод ветвей с отсечением [20];
- динамическое программирование [5].

Такие подходы перебирают решения, пока не будет найдено оптимальное. В худшем случае будут перебираться все варианты возможных решений.

Время работы точных алгоритмов на практике велико, поэтому чаще применяются эвристические и метаэвристические. В этом случае ищутся приближенные решения, которые находятся достаточно быстро и достаточно точны для требуемых целей.

В эвристических методах производится ограниченный поиск по пространству решений (т.е. нет перебора всевозможных вариантов). Для эвристического алгоритма известно, что он дает достаточно хорошие результаты для большинства случаев, но не доказана правильность для всевозможных случаев. Примерами таких алгоритмов являются:

- метод сбережений [21];
- метод, основанный на совпадениях [22].

В метаэвристических методах изучаются наиболее перспективные части пространства решений. Качество решений получается выше, чем у полученных классическими эвристиками. К этой группе относятся:

- метод поиска с запретами [10,11];
- генетические алгоритмы [17];
- муравьиный алгоритм [13].

Алгоритмы решения задачи транспортной маршрутизации можно разделить на следующие группы:

1. Конструктивные алгоритмы. Решение строят постепенно, отслеживая рост его стоимости. Нет фазы дальнейшего улучшения.
2. Двухфазные алгоритмы. При решении задача разделяется на две части: сначала вершины группируют по ТС, а затем составляют маршруты для каждого ТС.
3. Улучшающие алгоритмы. Сначала формируют допустимые решения, а затем делают попытки обмена вершин внутри каждого маршрута или между маршрутами.

2.1. Метод k-средних

Первым этапом в решении задач маршрутизации с несколькими транспортными средствами является распределение клиентов по ТС. Для этого необходимо воспользоваться методами кластерного анализа.

Кластеризация – задача разбиения на непересекающиеся подмножества (кластеры) заданной выборки объектов. Объекты в одном кластере близки по свойствам и значительно отличаются от объектов в других кластерах. Задача относится к классу задач обучения без учителя – задач обработки данных, в которых известны описания объектов и требуется определить взаимосвязи между объектами.

Наиболее известный способ решения – метод k-средних,

предложенный в 1950-х годах Г. Штейнгаузом [23]. Необходимо разделить m объектов из пространства R^n на k кластеров, при этом каждый объект относится к тому кластеру, к центру которого он ближе всего. Центры кластеров называются центроидами. В качестве меры близости используется Евклидово расстояние: $\rho(x,y) = ||x-y|| = \sqrt{\sum_{p=1}^{n}(x_p - y_p)^2} \quad x, y \epsilon R^n$.

Пусть дана выборка $x^{(1)} \dots x^{(m)} \epsilon R^n$. Метод k-средних разделяет эту выборку на k (k<m) групп ($y^1 \dots y^k$), при этом минимизируется суммарное квадратичное отклонение точек кластера от центроидов этих кластеров: $\min\left\{ \sum_{j=1}^{k} \sum_{x^i \epsilon y^j} \left||x^i - c^j|\right| \right\}$, $c^j -$ центроида кластера j.

Число k является параметром метода и подается методу на входе. Метод не предусматривает автоматического определения оптимального количества кластеров.

На рисунке 2.1 представлена блок-схема метода.

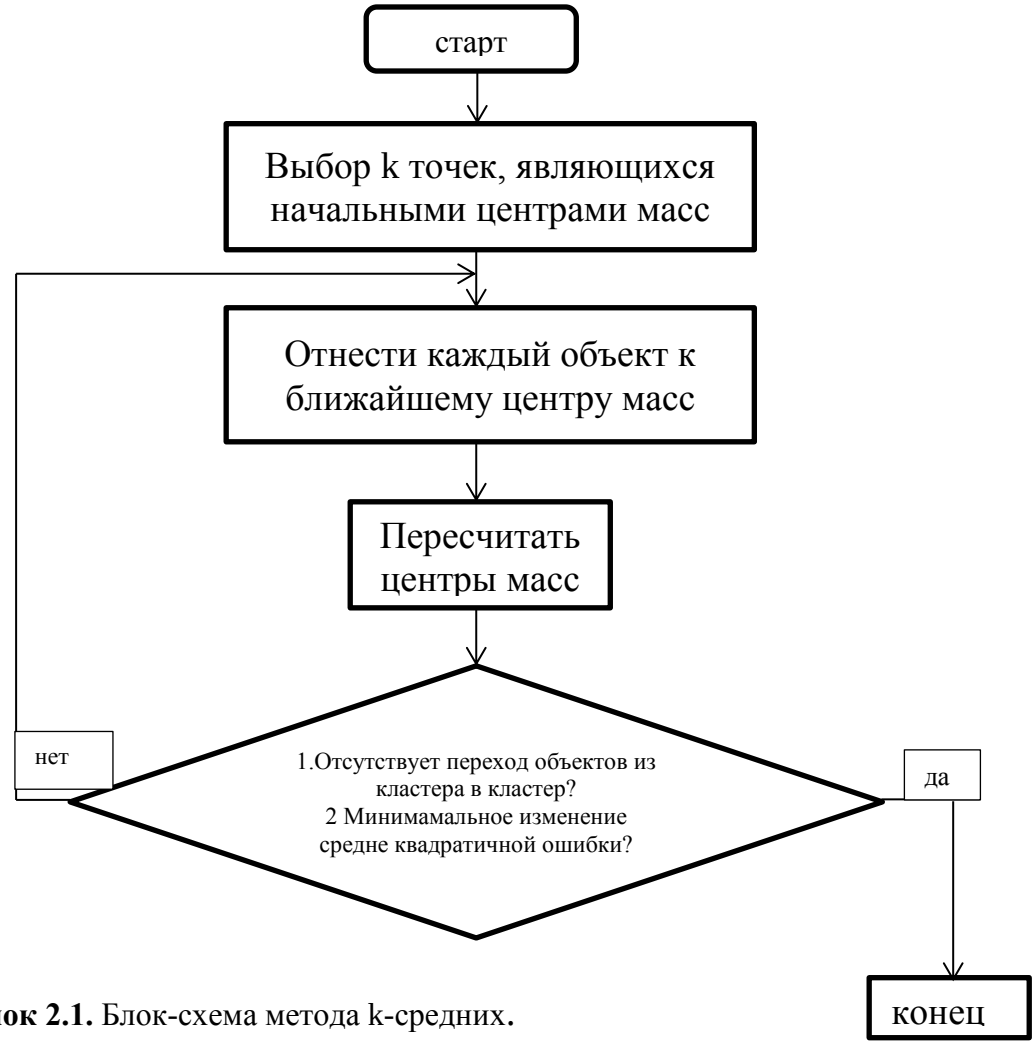

Рисунок 2.1. Блок-схема метода k-средних.

2.2. Метод поиска с запретами (Tabu Search)

Ф. Гловер в 1989 [24] предложил новую схему локального поиска. Она позволяет алгоритму перемещаться между локальными оптимумами в поисках глобального оптимума. Основным инструментом для этого является список запретов Tabu(i_k), строящийся на предыстории поиска (по нескольким предыдущим решениям) и запрещающий часть окрестности текущего решения N(i_k). Тогда на каждом шаге ищется оптимальное решение для подзадачи:

$$f(i_{k+1}) = \min\{ f(i) | \, i \epsilon N(i_k) \backslash Tabu(i_k)\}.$$

Список запретов запрещает использовать фрагмент решения (ребро графа, координат вектора), которые менялись на последних s шагах. Константа s является параметром метода, задающая длину списка Tabu. При $s = 1$ получается стандартный локальный поиск. Вторым параметром метода выступает мощность окрестности N(i_k) – p. При малом значении этого параметра метод не сможет перемещаться из одного локального оптимума в другой.

На рисунке 2.2 представлена блок схема метода:

f – значение целевой функции на некотором решении;

k – количество проделанных шагов;

fails – количество раз, когда алгоритм доходит до локального минимума без улучшения подряд.

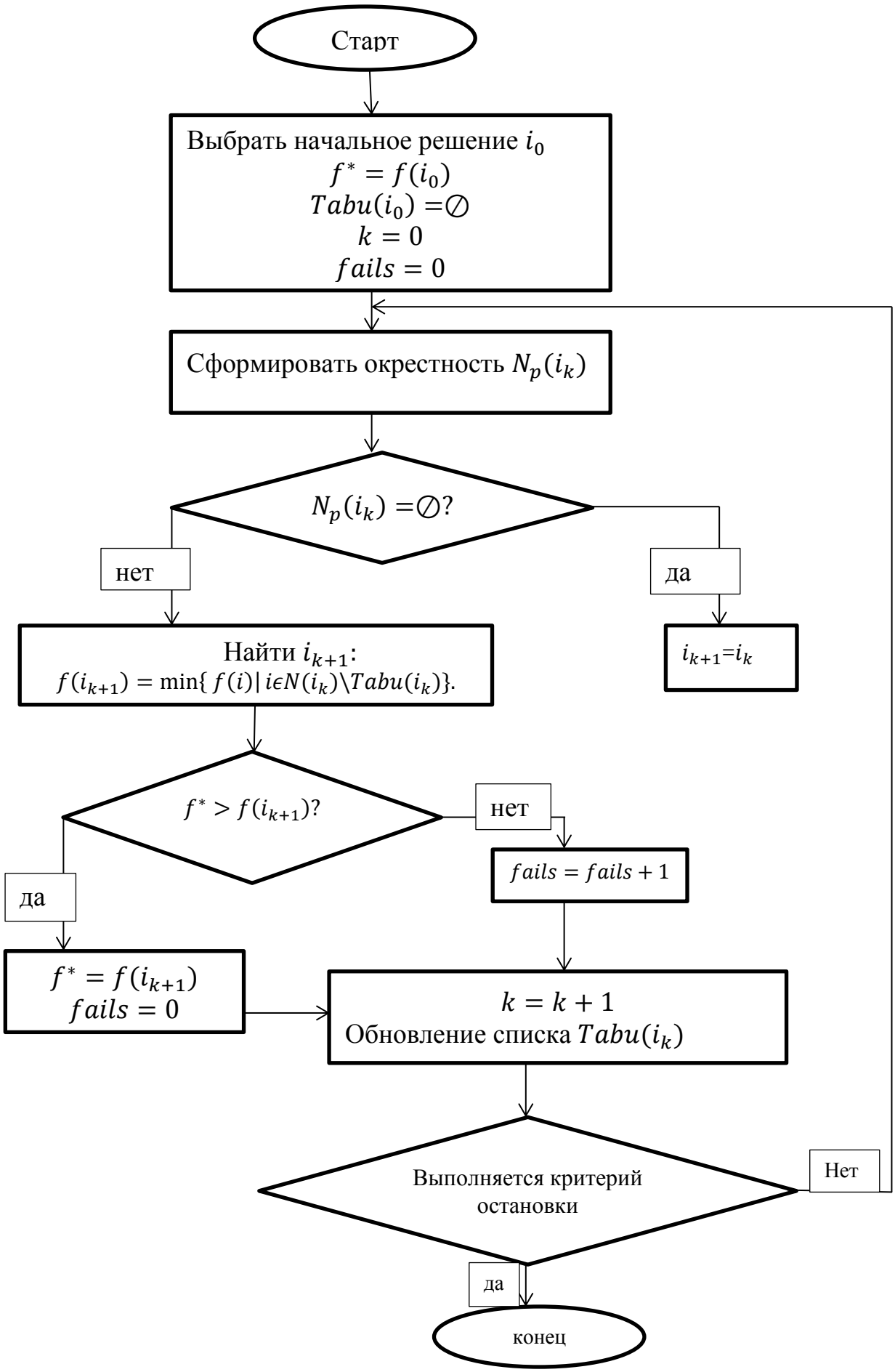

Рисунок 2.2. Блок-схема метода поиска с запретами

2.3. Генетический алгоритм (Genetic algorithm)

Генетический алгоритм – один из самых известных на сегодняшний день эвристических алгоритмов. В его основе лежат механизмы, аналогичные механизмам естественного отбора в природе. Идею метода предложил Дж. Холланд в конце 1960-х – начале 1970-х [25]. Его заинтересовали процессы естественной эволюции, свойства, которые решают в природе задачи совершенствования живых организмов: внутри популяции живых особей осуществляется отбор наиболее приспособленных и жизнестойких.

Для того, чтобы воспользоваться данным алгоритмом требуется описать искомое решение в виде вектора (хромосомы). Например, в рамках рассматриваемой задачи, это будет последовательность обхода вершин для ТС. Алгоритм имитирует эволюционные процессы в «поколениях» таких хромосом (поколение - набор фиксированного количества хромосом). Для поиска «хороших» хромосом алгоритму требуется оценка (fitness function), показывающая степень приспособленности для развития в нужном направлении хромосомы. В случае задачи VRP это будет функция, отображающая затраты на перевозку по маршруту, закодированного хромосомой. Таким образом, будет осуществляться выбор решений (оптимальных или близких к оптимальным) из множества альтернатив (популяций).

Алгоритм работает следующим образом. Для первого поколения хромосомы генерируются случайным образом. Далее начинается формирование новой популяции. С учетом оценки приспособленности выбираются решения, к которым применяются генетические операторы: скрещивание, мутация. Новые решения также получают оценку. Затем лучшие решения отбираются в новое поколение с тем же числом хромосом (этап селекции). Для нового поколения снова применяется тот же процесс. Формирование поколений происходит до тех пор, пока не будет выполнен критерий остановки алгоритма: нахождение глобального или

субоптимального решения или исчерпание числа поколений или времени, отпущенного на эволюцию. Общая схема метода представлена на рисунке 2.3.

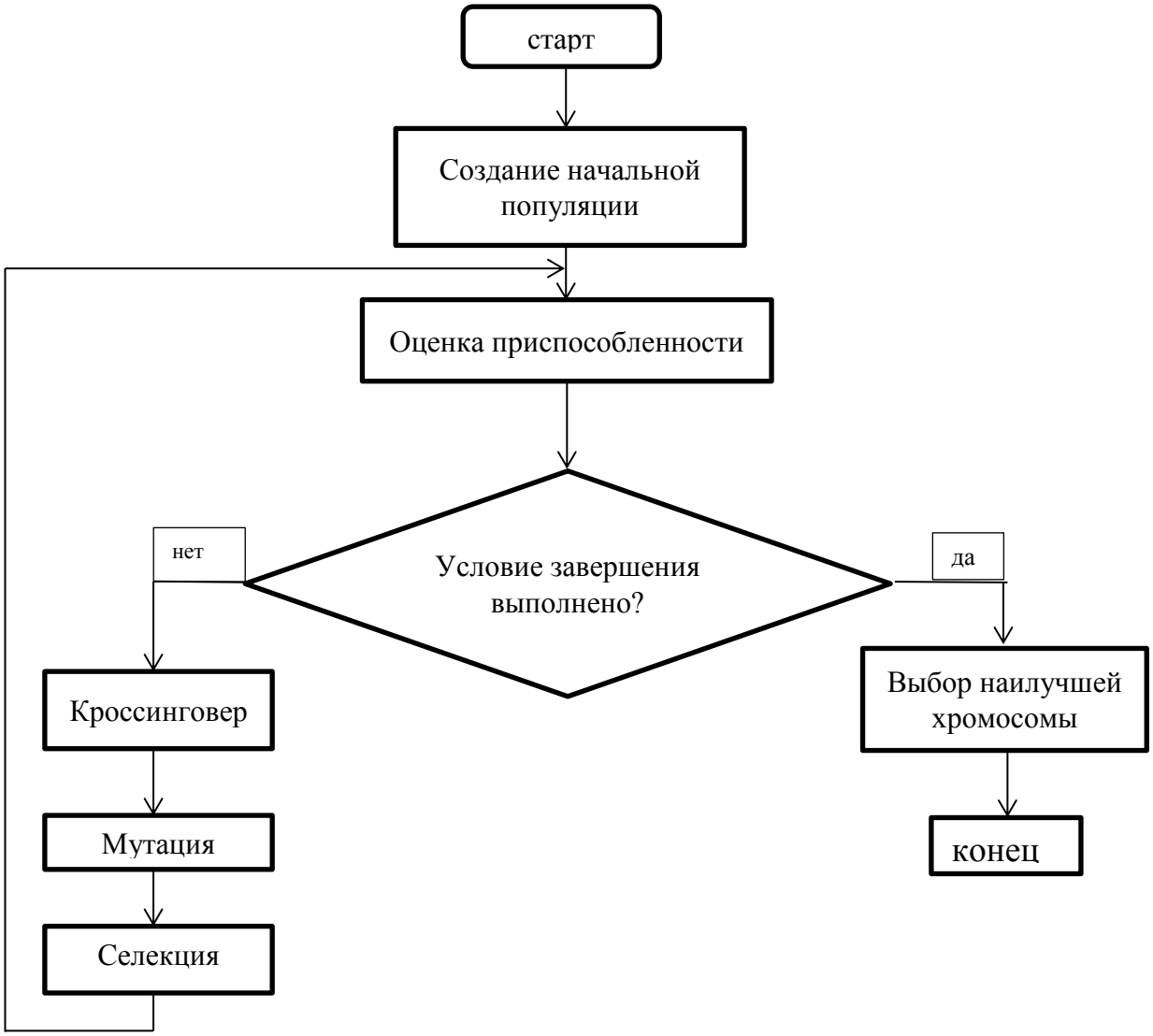

Рисунок 2.3. Блок-схема генетического алгоритма**.**

Глава 3. Динамическая адаптация метода

3.1. Временная состоятельность алгоритма

Одним из вариантов для оценки эффективности работы эвристического алгоритма является оценка временной состоятельности решений. Эвристические алгоритмы позволяют найти хорошее решение в большинстве случаев, но не гарантируют его оптимальность. А значит, нарушается принцип оптимальности Беллмана: «оптимальное решение обладает таким свойством, что независимо от начальных данных и начальной точки, оставшееся решение будет оптимальным, с учетом реализации первой точки» [26]. То есть на подзадачах оптимальное решение сохраняет оптимальность. Так как решение задач маршрутизации можно представить в виде последовательности вершин, которые посещает ТС, то к ним применима оценка временной состоятельности.

Пусть s – это решение, построенное эвристическим методом. Обозначим $l(s)$ – суммарные затраты на переезд. Весь маршрут s разделим на N этапов, переменная N фиксируется перед началом эксперимента. Рассмотрим некоторый этап n. Путь s можно представить как объединение двух частей: s^+ – часть маршрута, соответствующая порядку обхода пунктов после этапа n; s^- – оставшаяся часть.

Теперь рассмотрим новую задачу, исключив вершины множества s^- из исходной. Начальное депо для новой задачи – последняя вершина из s^-. А для задачи с несколькими ТС возникает задача с несколькими депо, так как новые расчеты проводятся для ситуации, когда ТС занимают разные текущие позиции. Воспользовавшись тем же алгоритмом, находим решение новой задачи s_n. Его стоимость $l(s_n)$.

Определение. Решение *динамически устойчивое*, или *состоятельное во времени*, если для всех $n = 1, \dots, N-1$ выполняется следующее неравенство

$$l(s^+) \leq l(s_n)$$

Рассмотрим множество тестовых примеров T. Тогда $t \in T$ – один из примеров, а $S(t)$ – множество решений, полученных для этого примера эвристическим алгоритмом. Для каждого решения проведем серию из M вычислительных экспериментов на временную состоятельность. Обозначим число экспериментов решения $s \in S(t)$ теста t, для которых это свойство нарушается на этапе n, через $b(s,t,n)$. Если решение оптимальное, то, согласно принципу оптимальности Беллмана, выполняется равенство $\sum_{n=1}^{N-1} b(s,t,n) = 0$.

Определение. *Экспериментальным уровнем временной состоятельности эвристического алгоритма* L называется величина, определяемая по формуле:

$$con\ L = 1 - \frac{1}{M|T|}\sum_{t \in T}\frac{1}{|S(t)|}b(s,t,n)$$

Таким образом, оценка временной состоятельности алгоритма определяется экспериментально как процент решений, которые оказались динамически устойчивыми. Коэффициент $con\ L$ принимает значение в диапазоне $[0,1]$. Чем ниже коэффициент динамической устойчивости, тем больше вероятность того, что начальное решение может быть улучшено.

3.2. Динамическая адаптация алгоритма

Воспользуемся идеей динамической адаптации эвристического алгоритма, подробно рассмотренной в статье [27] для модификации алгоритма решения рассматриваемой задачи.

Динамическая адаптация эвристического алгоритма состоит в следующем: на первом этапе генерируем решение с использованием алгоритма, выбранного для решения данной задачи, далее разбиваем маршруты на N частей. Фиксируем первую часть маршрута, а для оставшейся части снова запускаем тот же алгоритм и составляем решение. Если новое решение лучше предыдущего, то маршрут меняется. Далее зафиксируем еще и вторую часть, а для остальной части строим решение и в

случае улучшения, меняем маршрут. Повторяем действия до тех пор, пока не пройдем все части.

Динамическая адаптация часто позволяет найти решение с меньшим значением целевой функции. Это связанно с тем, что уменьшается размерность задачи, а эвристические алгоритмы более точны на задачах меньшей размерности. Кроме того, каждая подзадача отличается от исходной своей топологией, поэтому могут быть найдены другие глобальные или локальные оптимумы.

Таким образом, общая схема для поиска решения с помощью динамически адаптированного алгоритма для рассматриваемой задачи выглядит следующим образом:

1. Поиск решения с помощью эвристического алгоритма.
2. Запускаем модифицированный алгоритм:
    2.1. J=1.
    2.2. Фиксируем J частей.
    2.3. Для остальных вершин запускаем тот же эвристический алгоритм, формируем новые решения, J=J+1.
    2.4. Если J<N, то возвращаемся на шаг 2.2.

Глава 4. Решение задачи маршрутизации 1-1 вывоза и доставки товара с несколькими транспортными средствами одинаковой грузоподъемности

4.1. Постановка задачи

Для подробного рассмотрения выбрана статическая задача вывоза и доставки товара типа один к одному с несколькими ТС одинаковой грузоподъемности.

Имеется конечное множество поставщиков и потребителей. Каждому поставщику соответствует свой потребитель, поэтому количество поставщиков и потребителей одинаковое и равно n. Для решения задачи используется парк из k транспортных средств (ТС) одинаковой грузоподъемности, которую нельзя превышать. Все ТС начинают и заканчивают свой путь в депо (одно для всех ТС).

Перед выездом из депо имеется информация обо всех запросах на перевозку, координаты всех поставщиков и потребителей на плоскости.

Целью задачи является построение маршрутов для каждого ТС, при которых будут обслужены все поставщики и потребители. При этом минимизируются общие затраты на обслуживание всех клиентов. Затраты в данной задаче будут определяться расстоянием между вершинами.

4.2. Математическая модель задачи

Задача задается на ориентированном графе $G = (V, A)$, где $V = P \cup D \cup \{0, 2n + 1\}$ – множество вершин, A – множество дуг. Здесь $P = \{1, \dots, n\}$ – поставщики, $D = \{n + 1, \dots, 2n\}$ – потребители. Для каждого элемента $i \in P$ ставится в соответствие элемент $i + n \in D$. Выезд всех транспортных средств осуществляется из депо, сюда же они должны вернуться после окончания маршрута. Вершины $\{0\}$ и $\{2n + 1\}$ обозначают это депо. С каждой вершиной i ассоциировано некоторое количество груза q_i. Если $i \in P$, то $q_i \geq 0$, и если $i \in D$, то $q_i \leq 0$. При этом выполняется

равенство $q_i = -q_{i+n}$. Для вершин соответствующих депо: $q_0 = q_{2n+1} = 0$. В задаче рассматривается однородный груз. Накладываются ограничения на порядок обхода: поставщик должен быть обслужен раньше соответствующего потребителя, и каждый пункт посещается только один раз и только одним ТС.

Пусть (i, j) – путь, соединяющий вершины i и j, тогда $A = \{(i,j): (i = 0,\ j \in P)\ or\ (i,j\ \in V, i \neq j, i \neq n+j)\ or\ (i \in D, j = 2n+1)\}$. Каждая дуга характеризуется стоимостью переезда c_{ij}.

Для решения задачи, используется парк из $k \in K$ ТС одинаковой грузоподъемности Q. На любом этапе маршрута нельзя допустить переполнения ТС и перевозки отрицательного значения груза.

Целью задачи является построение замкнутых маршрутов для каждого ТС, при которых будут обслужены все поставщики и все потребители. При этом минимизируются общие затраты на обслуживание всех клиентов.

Для описания математической модели введем следующие переменные:

$X_{i,j}^k$ – бинарная переменная, принимает значение 1, если дуга (i, j) принадлежит маршруту k-го транспортного средства, 0 - в противном случае.

W_i^k – целочисленная переменная, показывающая количество свободного места в ТС с номером k после посещения вершины i.

Тогда задача представляется следующими соотношениями:

$$min \sum_{i \epsilon V} \sum_{j \epsilon V} \sum_{k \epsilon K} c_{ij}\, X_{i,j}^k \quad (1)$$

$$\sum_{k \in K} \sum_{j \in V} X_{i,j}^k = 1,\ \forall i \in P, \quad (2)$$

$$\sum_{i \epsilon V \setminus \{0\}} X_{0,i}^k = \sum_{i \in V\{2n+1\}} X_{i,2n+1}^k = 1,\ \forall k\ \in K, \quad (3)$$

$$\sum_{j \in V \setminus \{0\}} X_{i,j}^k = \sum_{j \in V \setminus \{2n+1\}} X_{j,i}^k,\ \ \forall i \in V,\ \forall k \in K, \quad (4)$$

$$\sum_{j \in P \cup D} X_{i,j}^k = \sum_{j \in V \setminus \{0\}} X_{i+n,j}^k,\ \forall i \in P,\ \forall k \in K, \quad (5)$$

$$\sum_{j \in V \setminus \{2n+1\}} X_{j,i}^k = \sum_{j \in P \cup D} X_{j,i+n}^k,\ \ \forall i \in P,\ \forall k \in K, \quad (6)$$

$$W_j^k \geq \left(W_i^k + q_j\right) X_{i,j}^k,\ \ \forall i,j \in V,\ \forall k \in K, \quad (7)$$

$$\max\{0, q_i\} \leq W_i^k \leq \min\{Q_k, Q_k + q_i\}, \ \forall i \in V, \forall k \in K, (8)$$
$$X_{i,j}^k \in \{0,1\}, \forall i, j \in V, \forall k \in K.$$

Здесь (1) — это минимизируемая функция, характеризующая сумму затрат по всем маршрутам. Условие (2) означает, что каждая вершина посещается только один раз. Условия (3) и (4) гарантируют, что каждое ТС начинает и заканчивает свой путь в депо. Ограничения (5), (6) — это условия предшествования: каждое ТС должно посетить вершину потребителя, если посещена соответствующая ему вершина поставщика, и каждое ТС должно покинуть вершину потребителя, если покинута соответствующая ему вершина поставщика. Ограничение (7) отражает изменение объема груза в ТС после переезда из вершины i в вершину j; (8) — ограничения на объем груза в ТС: $[q_i, Q]$ для вершины-поставщика и $[0, Q + q_i]$ для вершины-потребителя.

4.3. Эксперимент: решение тестовых задач из библиотеки NEO

Для проведения эксперимента были рассмотрены 7 тестовых наборов точек из библиотеки NEO [29]. Каждый тест состоит из 100 точек, заданных на координатной плоскости. Каждая вершина требует или поставляет груз объемом 1, соответственно у поставщиков +1, а у потребителей -1. По условию задачи каждому поставщику соответствует свой потребитель, поэтому в тестовых данных груз из вершины i доставляется в вершину $i + n$. Для каждого теста проводилась серия экспериментов с разным количеством транспортных средств и разной грузоподъемностью. На рисунке 5.1 подставлен формат тестового набора.

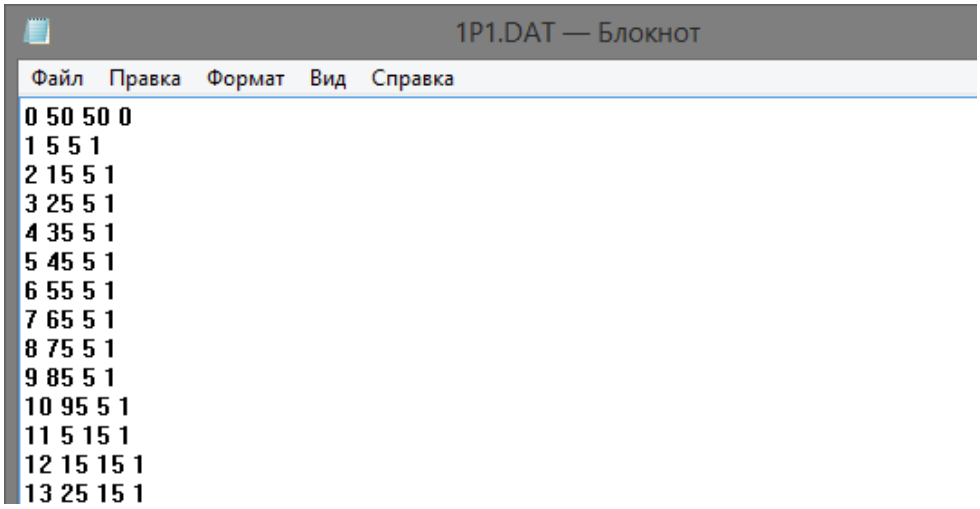

Рисунок 5.1. Фрагмент файла для тестовой задачи 1P1.DAT из библиотеки NEO.

Решение задачи маршрутизации с несколькими ТС разбивается на два этапа:

1. Разбиение вершин по ТС;
2. Построение маршрутов для каждого ТС.

На первом этапе воспользуемся методом кластерного анализа, а именно, методом k-средних, подробно описанным в пункте 2.1 главы 2.

На втором этапе формируем маршруты метаэвристическими методами: метод писка с запретами (пункт 2.2 главы 2) и генетический алгоритм (пункт 2.3 главы 2) .

4.4. Сравнение метода поиска с запретами и генетического алгоритма

Алгоритмы были реализованы на языке программирования Python. В *приложении 1* представлен метод k-средних, в *приложении 2* – метод поиска с запретами, а в *приложении 3* основные функции генетического алгоритма. Кроме того, оба метода требуют на входе некоторого начального решения. Для поиска этого решения, был реализован жадный алгоритм (*приложение 4*).

Каждым методом были найдены решения для 77 тестов. На каждом тесте алгоритмы запускались по 20 раз. В таблице 1 представлены результаты для одного из тестовых наборов точек. Результаты для всех

тестов в *приложении 6.*

k	Q	Лучшее решение		Среднее решение	
		TS	GA	TS	GA
3	30	870,46	869,09	881,51	870,87
3	15	1071,52	1103,65	1101,48	1104,76
3	10	1375,21	1284,44	1393,41	1292,46
3	5	2189,66	1657,99	2209,38	1688,65
5	30	1275,81	1316,79	1275,91	1319,25
5	10	1440,54	1304,13	1440,54	1308,89
5	5	2428,91	1824,12	2430,01	1828,86
5	3	2844,22	2620,05	2845,32	2714,78
10	30	2116,21	1978,31	2116,21	1982,15
10	5	2422,77	2450,33	2453,18	2456,19
10	3	3312,45	3194,32	3349,81	3305,31

Таблица 1. Результаты работы алгоритмов для теста 21P1.DAT

Здесь k – количество машин, Q– ограничение на грузоподъемность. Во втором столбце представлены лучшие значения целевой функции, полученные для данного случая методом поиска с запретами (TS) и генетическим алгоритмом (GA). В последнем столбце расположены средние значения целевой функции, полученные при применении метода поиска с запретами (TS) и генетического алгоритма (GA). Из таблицы видно, что были случаи, когда TS давал лучшее значение целевой функции меньше чем GA (k = 3, Q = 15), но таких случаев немного и разница 2-3%. А также были ситуации, в которых GA позволял найти решение с лучшим значением целевой функции на 10 % меньше чем TS (k = 5, Q = 10). Если же оценивать среднее значение целевой функции, то результаты для GA всегда ниже, чем для TS, или почти равны (отличие на 0,2-0,5%). В целом, по результатам всех тестов, можно сделать вывод о том, что GA в среднем генерирует решения со значением целевой функции на 7% меньше, чем TS.

Если сравнить алгоритмы по времени поиска решения, то TS затрачивает меньше времени на поиск. Среднее время работы TS для поиска

решения для одного теста составляет 5,819 секунд, и для всех тестов нет сильного разброса по времени. А для GA среднее время – 41,241. И время работы алгоритма зависит от количества машин и грузоподъемности. Для тестов с 3 ТС и большим ограничением на грузоподъемность (Q = 3, Q = 5) время работы алгоритма доходит до 5 минут.

Таким образом, метод поиска с запретами выигрывает по времени поиска решения генетический алгоритм, но уступает по качеству найденного решения.

4.5. Динамическая адаптация метода поиска с запретами

Эксперимент из 77 тестов, запускаемых по 20 раз, показал, что значение экспериментального уровня временной состоятельности метода поиска запретами для рассматриваемой задачи равен 0,31. Данное значение является довольно низким: только треть сгенерированных на начальном этапе решений сохраняет свойство оптимальности. А значит, существуют другие маршруты, значение целевой функции которых будет меньше. Их можно получить динамически с помощью того же метода.

Воспользуемся идеей динамической адаптации алгоритма описанной в главе 4 для модификации метода поиска с запретами для рассматриваемой задачи. Общая схема для поиска решения выглядит следующим образом:

1. Ввод данных.

2. Запускаем метод k-средних, распределяем пункты по машинам.

3. Для каждой группы запускаем жадный алгоритм, составляем множество начальных маршрутов.

4. Для каждой группы запускаем TS, ищем наилучшие решения.

5. Для каждой группы запускаем модифицированный алгоритм TS:

5.1. J=1.

5.2. Фиксируем J частей.

5.3. Для остальных вершин запускаем метод запретов,

формируем новые решения, J=J+1.

5.4. Если J<N, то возвращаемся на шаг 5.2.

Алгоритм был реализован на языке программирования Python и были найдены результаты тех же тестов. Код программы в *приложении 5*. В таблице 2 приведены результаты для одного из тестов (полные результаты в *приложении 6*).

k	Q	Лучшее решение		Среднее решение	
		TS	DTS	TS	DTS
3	30	870,46	865,86	881,51	872,69
3	15	1071,52	1028,66	1101,48	1053,24
3	10	1375,21	1279,32	1393,41	1289,16
3	5	2189,66	1630,09	2209,38	1676,58
5	30	1275,81	1275,81	1275,91	1275,81
5	10	1440,54	1440,43	1440,54	1440,43
5	5	2428,91	1593,39	2430,01	1594,49
5	3	2844,22	2567,89	2845,32	2568,99
10	30	2116,21	2116,16	2116,21	2116,16
10	5	2422,77	2422,77	2453,18	2434,87
10	3	3312,45	3201,07	3349,81	3212,15

Таблица 2. Результаты работы алгоритмов для теста 21P1.DAT

Здесь k – количество машин, Q – ограничение на грузоподъемность. Второй столбец – лучшее значение целевой функции, полученное для данного случая простым методом поиска с запретами (TS) и динамически адаптированным методом (DTS). В последнем столбце расположены средние значения целевой функции при применении обычного (TS) и модифицированного (DTS) методов. Из таблицы видно, что были случаи, когда модификация не приводила к уменьшению целевой функции (k = 10, Q = 5), а также были ситуации, в которых происходило уменьшение на 20-30% (k = 3, Q = 5). В целом, по результатам всех тестов, можно сделать вывод о том, что динамическая адаптация в большинстве случаев дает уменьшение стоимости переездов по маршрутам до 5%.

4.6. Сравнение динамически адаптированного метода поиска с запретами и генетического алгоритма

После модификации метод поиска с запретами стал находить решения с меньшим значением целевой функции. Сравним новые результаты с полученными ранее результатами для генетического алгоритма. В таблице 3 представлены результаты для одного из тестов (полные результаты в *приложении 6*).

k	Q	Лучшее решение		Среднее решение	
		DTS	GA	DTS	GA
3	30	865,86	869,09	872,69	870,87
3	15	1028,66	1103,65	1053,24	1104,76
3	10	1279,32	1284,44	1289,16	1292,46
3	5	1630,09	1657,99	1676,58	1688,65
5	30	1275,81	1316,79	1275,81	1319,25
5	10	1440,43	1404,13	1440,43	1408,89
5	5	1593,39	1824,12	1594,49	1828,86
5	3	2567,89	2620,05	2568,99	2714,78
10	30	2116,16	1978,31	2116,16	1982,15
10	5	2422,77	2450,33	2434,87	2456,19
10	3	3201,07	3194,32	3212,15	3305,31

Таблица 3. Результаты работы алгоритмов для теста 21P1.DAT

Здесь k – количество машин, Q – ограничение на грузоподъемность. Второй столбец – лучшее значение целевой функции, полученное для данного случая динамически адаптированным методом поиска с запретами (DTS) и генетическим алгоритмом (GA). В последнем столбце расположены средние значения целевой функции для DTS и GA.

Анализируя случаи в таблице, видим, что при Q = 30 алгоритм GA генерирует решения с меньшим значением целевой функции. Самая большая разница 6% в случае k = 10, Q = 30. В остальных случаях DTS позволяет найти решения с меньшим значением целевой функции, разница до 2% (k = 3, Q = 10), а где-то доходит до 14% (k = 5, Q = 5). Сопоставляя с

результатами из таблицы 1, видим, что модификация метода с запретами позволила уменьшить стоимость получаемых решений, особенно в тех случаях, где была большая разница с решениями, получаемыми GA. Например, в случае k = 3, Q = 5 простой метод поиска с запретами генерировал решения со значением целевой функции на 20% больше, чем генетический алгоритм, а модифицированный метод генерирует решение со средним значением целевой функции, практически совпадающим со значением для решения, полученного генетическим алгоритмом.

Из результатов всех тестов можно сделать следующие выводы:

1) GA при больших Q (Q = 15, Q = 10) генерирует решения со значением целевой функции в среднем на 6% меньше, чем у решений, полученных DTS (наибольшая разница доходит до 17%).
2) При наименьшем значении Q (5 или 3 в зависимости от количества машин) алгоритмы генерируют близкие по стоимости решения: отличие в среднем на 2%.
3) Для остальных значений Q после модификации средние решения DTS и GA приблизились друг к другу, особенно в тех ситуациях, где динамическая адаптация улучшила решение больше, чем на 10%.

По времени работы генетический алгоритм уступает модифицированному методу поиска с запретами. Среднее время работы DTS равно 8,187 с небольшим разбросом по времени между тестами, а для GA 41,241 – среднее время поиска решения и для некоторых тестов время вычисления доходит до 5 минут.

Таким образом, модификация метода поиска с запретами на тестах с сильными ограничениями на грузоподъемность позволяет получить результаты как при генетическом алгоритме, значительно сэкономив на времени. А на тестах с большим значением Q генетический алгоритм по-прежнему имеет преимущество.

Выводы

Для задачи маршрутизации с вывозом и доставкой с несколькими транспортными средствами ограниченной грузоподъемности на языке программирования Python был реализован двухэтапный подход решения задачи. Для первого этапа – метод k-средних, для второго – генетический алгоритм, метод поиска с запретами и его модификация с помощью идеи динамической адаптации. Таблица с результатами дает основания сделать следующие выводы:

1. Генетический алгоритм позволяет генерировать решения с меньшим значением целевой функции, чем метод поиска с запретами.
2. Использование динамически адаптированного метода поиска с запретами улучшает решение, полученное простым методом поиска с запретами, в большинстве случаев до 5%.
3. Для случаев с большим значением грузоподъемности ТС генетический алгоритм генерирует маршруты с меньшей стоимостью, чем метод поиска с запретами и его модификация.
4. Для остальных случаев модификация метода поиска с запретами генерирует решения близкие к генетическому алгоритму, но за более короткое время.

Заключение

В данной работе был рассмотрен класс задач транспортной маршрутизации и подробно исследована одна из задач - задача маршрутизации с вывозом и доставкой с несколькими транспортными средствами ограниченной грузоподъемности.

Были изучены подходы к решению данного типа задач. Был выбран двухэтапный метод решения задач: на первом этапе было распределение вершин по ТС, а на втором - формирование маршрутов для каждого ТС. На языке программирования Python был реализован метод k-средних для первого этапа. Для второго были выбраны метод поиска с запретами и генетический алгоритм, реализованные на том же языке. Далее на тестовых примерах произведено сравнение результатов алгоритмов: генетический алгоритм генерирует решения с меньшим значением целевой функции, в среднем на 7%, но за более длительное время.

Далее экспериментально определена оценка временной состоятельности метода поиска с запретами. Эксперимент показал, что всего треть решений являются динамически устойчивыми, а значит, решения могут быть улучшены. Поэтому далее метод модифицирован с помощью идеи динамической адаптации алгоритмов. Анализ экспериментов показал, что улучшение решения на тестах с различным количеством ТС и различной грузоподъемностью в большинстве случаев доходит минимум до 5%. Таким образом, использование на практике динамически адаптированного метода поиска с запретами позволяет генерировать маршруты меньшей стоимости.

На последнем этапы было произведено сравнение модифицированного метода поиска с запретами и генетического алгоритма. Результаты экспериментов позволили определить случаи, когда лучше использовать генетический алгоритм, а когда преимущество имеет модифицированный метод поиска с запретами.

Литература

1. Dantzig G. B., Ramser J. H. The Truck Dispatching Problem // Management Science. 1959. Vol. 6, No 1. P. 80–91.
2. Berbeglia G., Cordeau J., Laporte G. Dynamic pickup and delivery problems // European journal of operational research. 2010. No 202. P. 8-15.
3. Berbeglia G., Cordeau J., Gribkovskaia I., Laporte G. Static pickup and delivery problems: classification scheme and survey // Springer. 2007. No 15. P. 1-31.
4. Psaraftis Harilaos N. Analysis of an $O(N^2)$ heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem // Transpn Res.-B. 1983. Vol. 17B, No 2. P. 133–145.
5. Harilaos N. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem // Transportation science. 1980. Vol. 14. No. 2. P. 385-392.
6. Gribkovskaia I., Laporte G. One-to-one Single Vehicle Pickup and Delivery Problems // Single Vehicle Pickup and Delivery Problems. 2008. P. 359-377.
7. Cordeau J., Laporte G., Ropke S. Recent Models and Algorithms for One-to-One Pickup and Delivery Problems // The vehicle routing problem: latest advances and new challenges. 2008 P. 329-359
8. Hernandez-Perez H., Salazar-Gonzalez J. The multi-commodity one-to-one pickup-and-delivery traveling salesmen problem // European journal of operation research. 2009. Vol. 196. P. 987-995.
9. Anily S, Hassin The swapping problem // Networks. 1992. No. 22. P. 419-433.
10. Gribkovskaia I., Laporte G., Shlopak A. A tabu search heuristic for a routing problem arising in servicing of offshore oil and gas platforms // Journal of the operational research society. 2008. No 59. P. 1449-1459.
11. Cordeau J., Laporte G. A tabu search heuristic for the static multi-vehicle

dial-a-ride problem // Transportation research. 2003. Part B. Vol. 37. P.579-594.

12. Powell W.B., Bouzanene-Ayari B., Simpo H.P. Dynamic models for freight transportation // Handbooks in operation research and management science: transportation. 2007. Vol. 14. P.285-365.
13. Abdulkader M., Gajpal Y., ElMekkawy T. Hybridized ant colony algorithm for the Multi Compartment Vehicle // Applied Soft Computing. 2015. No 37. P. 196–203.
14. Grandinetti L., Guerrieo F., Pezzella F., Pisacane O. The multi-objective multi-vehicle pickups and delivery problem with time windows // Procedia-social and behavioral sciences. 2014. Vol. 111. P. 203-212.
15. Nagy G., Salhi S. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries // European Journal of Operational Research. 2005. Vol. 162. No. 1. P. 126-141.
16. Dror M., Laporte G., Trudeau P. Vehicle routing with split deliveries // Discrete applied mathematics. 1994. Vol. 50. No. 3. P.239-254.
17. Vidal T., Crainic T.C., Gendreau M. A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems // Operations Research. 2012. Vol. 60. No. 3. P. 721-738.
18. Heilporn G., Cordeau J.-F., Laporte G. An Integer L-shaped Algorithm for the Dial-a-Ride Problem with Stochastic Customer Delays // CIRRELT. 2010. No.29.
19. Hernandez-Perez H., Salazar-Gonzalez J. A branch-and-cut algorithm for a traveling salesmen problem with pickup and delivery // Discreet applied mathematics. 2004. Vol. 145. P. 126-139.
20. Hu T.-Y., Chang C.-P. A revised branch-and-price algorithm for dial-a-ride problems with consideration of time-dependent travel cost // Journal of advanced transportation. 2015. Vol. 49. P.700-723.
21. Corominas A., Garcia-Villoria A., Pastor R. Improving parametric Clarke and Wright algorithms by means of iterative empirically adjusted greedy

heuristics // SORT. 2014. No 1. P. 3-12.

22. Desrochers M., Verhoog T.W. A Matching Based Savings Algorithm for the Vehicle Routing Problem // GERAD. 1989. No 4. P. 1-19.
23. Steinhaus H. Sur la division des corp materiela en parties // Bull Acd. Polon. Sci. 1956. C1 III Vol IV P. 801-804.
24. Glover F. Tabu search – Part I // ORSA Journal on Computing. 1989. No. 1. P. 190-206.
25. Holland J.H. Adaptation in natural and artificial systems (1975) // The university of Michigan press. 1992.
26. Беллман Р. Динамическое программирование // Изд-во иностранной литературы. 1960. 400 С.
27. Shirokikh V.A., Zakharov V.V. Dynamic adaptive large neighborhood search for inventory routing problem // Advances in intelligent systems and computing. 2015. Vol. 359. P. 231-243.
28. Захаров В.В. Методы и модели прикладной математической логики // Процессы управления и устойчивость. 2015. Т. 2(18). С. 742-776.
29. Networking and emerging optimization [Электронный ресурс]: URL: http://neo.lcc.uma.es/vrp/ (дата обращения: 15.01.2019).
30. Трифонов Ю.В., Громницкий В.С., Золотов М.Ю. Формирование оптимальных маршрутов доставки товаров автотранспортом // Вестник Нижегородского университета им. Н.И.Лобачевского. 2010. № 6. С. 236-240.
31. Ишкова Е.С. Матричный метод решения задачи маршрутизации с несколькими транспортными средствами с учетом ограничений на массу и объем перевозимого груза // Вестник ИНЖЭКОНА серия: Экономика. 2011. № 1(44). С. 329-333.
32. Şahın M., Çavuşlar G., Öncan T., Şahın G., Aksu D. An efficient heuristic for the Multi-vehicle One-to-one Pickup and Delivery Problem with Split Loads // Transportation Research. 2003. Part B. No. 37. P. 579-594.

Приложение

Приложение 1. Функция метода k-средних

```
k=3 #количество кластеров
centroids = vq.kmeans(np.array(data), k, iter=200)[0]
from scipy.spatial.distance import cdist
def kmeans_export(centroids, data, labels):
    h=[1 for i in labels]
    res = [[] for i in range(len(centroids))]
    d = cdist(np.array(data), centroids, 'euclidean')
    for i, l in enumerate(d):
        res[l.tolist().index(l.min())].append((labels[i], data[i]))
        h[labels[i]-1]=l.tolist().index(l.min()) #вершина - кластер
    return res,h
K_res,KL_res = kmeans_export(centroids, data, names)
res = [[] for i in range(len(centroids))]
num=0
for i in K_res:
    num=num+1
    for j in i:
        if j[0]<=n:
  k1=cdist(np.array(j[1]).reshape((1,2)),centroids[KL_res[j[0]- 1]].reshape((1,2)),'euclidean')
k2=cdist(np.array(j[1]).reshape((1,2)),centroids[KL_res[j[0]+n-1]].reshape((1,2)),'euclidean'
k3=cdist(np.array(data[j[0]+n-1]).reshape((1,2)),centroids[KL_res[j[0]-
1]].reshape((1,2)),'euclidean')
 k4=cdist(np.array(data[j[0]+n-1]).reshape((1,2)),centroids[KL_res[j[0]+n-
1]].reshape((1,2)),'euclidean')
            if k1+k3<=k2+k4:
                res[KL_res[j[0]-1]].append((j[0],j[1]))
                res[KL_res[j[0]-1]].append((j[0]+n,data[j[0]+n-1]))
            else:
                res[KL_res[j[0]+n-1]].append((j[0],j[1]))
                res[KL_res[j[0]+n-1]].append((j[0]+n,data[j[0]+n-1]))
```

Приложение 2. Функция метода поиска с запретами

```
def tabu_search(res,group,l,n1,D,queries,Q,rtabu):
    fail=0 n=len(group) rout=[] tabu=[] sum1=0
    for i in range(0,2*n1+2):
        rout.append([0]*(2*n1+2)
    for j in range(1,len(res)):
        rout[res[j-1]][res[j]]=1
        sum1=sum1+D[res[j-1]][res[j]]
    now=1 P1=[] P=[]
    for k in range(0,n):
        if group[k]<n1+1:
            P1.append(group[k])
    P1.remove(res[now])
    dnamber=res.index(res[now]+n1)
    for g in P1:
        if res.index(g)<dnamber:
            P.append(g)
    if len(P)==0:
        for k in range(0,n):
            if group[k]<n1+1:
                P.append(group[k])
        P.remove(res[now])
        now=2
    market=0 nowh=0 d=0
    while fail<l:
        if len(res)==4:
            fail=l
            break
        if now==n:
            fail=l
        else:
            res1=[]  W=[0]
            for jk in res:
                if (jk!=res[now]) and (jk!=P[nowh]):
                    res1.append(jk)
                else:
                    if res[nowh]>n1:
                        dnamber=res.index(res[nowh]-n1)
                        if dnamber<now:
                            if jk==res[now]:
                                res1.append(P[nowh])
                            else:
                                res1.append(res[now])
                        else:
```

```
                res1.append(jk)
        else:
            if jk==res[now]:
                res1.append(P[nowh])
            else:
                res1.append(res[now])
for jh in range(1,n+2):
    W.append(W[jh-1]+queries[res1[jh]])
rout1=[]
for ig in range(0,2*n1+2):
    rout1.append([0]*(2*n1+2))
sum2=0
for jf in range(1,len(res1)):
    rout1[res1[jf-1]][res1[jf]]=1
    sum2=sum2+D[res1[jf-1]][res1[jf]]
testsum=0
for h in W:
    if h>Q:
        testsum=1
if testsum>0:
    nowh=nowh+1
    if nowh<len(P):
        del P[nowh]
    while tabu.count(nowh)!=0:
        nowh=nowh+1
        if nowh>=len(P):
            break
        del P[nowh]
    if nowh>=len(P):
        now=now+1 nowh=0 P=[]
        for k in range(0,n):
            P.append(res[k])
        for k in range(0,now+1):
            if P.count(res[k])>0:
                P.remove(res[k])
        for k in P:
            if k<n1+1:
                if P.count(k+n1)>0:
                    P.remove(k+n1)
            if res[now]<n1+1:
                dnamber=res.index(res[now])   P1=[]
                for g in P:
                    if res.index(g)<dnamber:
                        P1.append(g)
                if len(P1)==0:
```

```
                now=now+1 P=[]
                for k in range(0,n):
                    P.append(res[k])
                for k in range(0,now+1):
                    if P.count(res[k])>0:
                        P.remove(res[k])
                for k in P:
                    if k<n1+1:
                        if P.count(k+n1)>0:
                            P.remove(k+n1)
            else:
                P=[]
                for t in P1:
                    P.append(t)
    if nowh>=len(P):
        fail=l
else:
    if sum2>=sum1:
        nowh=nowh+1
        if nowh<len(P):
            del P[nowh]
        while tabu.count(nowh)!=0:
            nowh=nowh+1
            if nowh>=len(P):
                break
            del P[nowh]
        fail=fail+1
        if nowh>=len(P):
            now=now+1 nowh=0
        P=[]
        for k in range(0,n):
            P.append(res[k])
        for k in range(0,now+1):
            if P.count(res[k])>0:
                P.remove(res[k])
        for k in P:
            if k<n1+1:
                if P.count(k+n1)>0:
                    P.remove(k+n1)
            if res[now]<n1+1:
                dnamber=res.index(res[now]) P1=[]
                for g in P:
                    if res.index(g)<dnamber:
                        P1.append(g)
                if len(P1)==0:
```

```
                now=now+1   P=[]
                for k in range(0,n):
                    P.append(res[k])
                for k in range(0,now+1):
                    if P.count(res[k])>0:
                        P.remove(res[k])
                for k in P:
                    if k<n1+1:
                        if P.count(k+n1)>0:
                            P.remove(k+n1)
            else:
                P=[]
                for t in P1:
                    P.append(t)
    if nowh>=len(P):
        fail=l
else:
    fail=0
    for lk in range(0,n):
        res[lk]=res1[lk]
    rout=[]
    for ip in range(0,2*n1+2):
        rout.append([0]*(2*n1+2))
    sum1=sum2
    for jp in range(1,len(res)):
        rout[res[jp-1]][res[jp]]=1
    tabu.append(res[now])
    tabu.append(P[nowh])
    now=1
    nowh=0
    P=[]
    for k in range(1,n):
        if res[k]<n1+1:
            P.append(res[k])
```

Приложение 3. Основные функции генетического алгоритма

```
def  generate_vector(center,n1,D,queries,Q):
    f=len(center)
    namber=[ i+1 for i in range(len(center))]
    new=random.sample(center,len(center))
    new1=first_solution(n1,D,queries,new,Q)
    center=[]
    for i in range(len(new1)):
        center.append(new1[i])
    return center
def  mutate(vect):
    _mutation = vect[:]
    n1=int((_mutation[len(_mutation)-1]-1)/2)
    rezer1=_mutation[0]
    rezer2=_mutation[len(_mutation)-1]
    del _mutation[0]
    del _mutation[len(_mutation)-1]
    fix=0
    n=len(_mutation)
    while fix!=1:
        s1=random.randint(0, len(_mutation)-2)
        P=[]
        for k in range(0,n):
            P.append(_mutation[k])
        for k in range(0,s1):
            if P.count(_mutation[k])>0:
                P.remove(_mutation[k])
        for k in P:
            if k<n1+1:
                if P.count(k+n1)>0:
                    P.remove(k+n1)
            if _mutation[s1]<n1+1:
                P1=[]
                dnamber=_mutation.index(_mutation[s1])
                for g in P:
                    if _mutation.index(g)<dnamber:
                        P1.append(g)
                if len(P1)==0:
                    s1=s1+1
                    if s1>=n:
                        break
                    P=[]
                    for k in range(0,n):
                        P.append(_mutation[k])
```

```
                for k in range(0,s1+1):
                    if P.count(_mutation[k])>0:
                        P.remove(_mutation[k])
                for k in P:
                    if k<n1+1:
                        if P.count(k+n1)>0:
                            P.remove(k+n1)
            else:
                P=[]
                for t in P1:
                    P.append(t)
        if len(P)>0:
            s2=random.randint(0, len(P)-1)
            si=_mutation.index(P[s2])
            s=_mutation[si]
            _mutation[si]=_mutation[s1]
            _mutation[s1]=s
        fix=1
    _mutation.append(rezer2)
    _mutation.insert(0,rezer1)
    return _mutation
def  mutation(generation, func):
    generation.sort(key=func)
    half = int(len(generation) // 2)
    mutated,weak = generation[:half], generation[half:]
    mutated.extend(map(mutate, weak))
    return mutated
def  cross_two(a, b, queries,n1,Q):
    cross=[]
    h=1
    a2=[]
    for i in a:
        a2.append(i)
    b2=[]
    for j in b:
        b2.append(j)
    del a2[0]
    del b2[0]
    sr=a2[len(a2)-1]
    del a2[len(a2)-1]
    del b2[len(b2)-1]
    for h in range(1,len(a2)-2):
        a1=a2[h:]
        b1=[]
        for kl in b2:
```

```
            b1.append(kl)
        for j in a1:
            b1.remove(j)
        c1=a1+b1
        fix=0
        for i in range(1,len(c1)-1):
            if (c1[i]<=n1) and (c1[i]>0):
                k=c1.index(c1[i]+n1)
                if k<i:
                    fix=fix+1
        if fix==1:
            for i in range(1,len(c1)-1):
                if (c1[i]<=n1) and (c1[i]>0):
                    k=c1.index(c1[i]+n1)
                    if k<i:
                        s=c1[k]
                        c1[k]=c1[i]
                        c1[i]=s
            fix=0
        if fix==0:
            c1.append(sr)
            c1.insert(0,0)
            cross.append(c1)
    return cross
def  crossover(generation, func, queries,n1,Q):
    generation.sort(key=func)
    half = int(len(generation) // 2)
    strong, weak  = generation[:half], generation[half:]
    for i in range(half):
        children = cross_two(strong[i], strong[-i],queries,n1,Q)
        strong.extend(children)
    return strong
def  disaster(generation, func, count,n3,D,queries,Q, pop_size=None):
    generation.sort(key=func)
    n1=pop_size
    if pop_size is int:
        n1 = pop_size
    elif pop_size is float:
        n1 = int(len(generation) * pop_size)
    else:
        n1 = int(len(generation) // 2)
    survived, weak = generation[:n1], generation[n1:]
    survived1=[]
    for i in range(len(survived)):
        survived1.append([])
```

```
        for j in range(len(survived[i])):
            survived1[i].append(survived[i][j])
    x = survived[:]
    last=0
    for i, item in enumerate(x):
        last=i
        survived.extend([[x for x in generate_vector(item,n3,D,queries,Q)] for _ in range(10)])
    for i in survived1:
        survived.append(i)
    survived.sort(key=func)
    return survived[:count]
def find_best(X, func):
    maxi = 0
    maxx = func(X[0][:])
    for i, x in enumerate(X):
        r = func(x)
        if r < maxx:
            maxi = i
            maxx = r
    return X[maxi]
class IterPrinter(object):
    def __init__(self, best,func, c=0):
        self.c = c
        self.best = best
        self.next(0, best,func)
    def next(self, i, best,func):
        self.c += i
        self.best = best
        return i
    def stop(self):
        print('\n')
def  norma(X, Y):
    return math.sqrt(sum([(x-Y[i])*(x-Y[i]) for i, x in enumerate(X)]))
def  _get_center(generation,func):
    n1=len(generation)
    fg=[]
    s=0
    g=0
    for i in range(len(generation)):
        g=func(generation[i])
        s+=g
        fg.append(g)
    return s/n1,fg
def  is_ideal(func, generation, eps):
    center,fg = _get_center(generation,func)
```

```
    return all(map(lambda y: abs(y-center) < eps, fg))
def simulate_evolution(func, first_generation, eps,Q,queries,pairs,D):
    evolution = []
    if type(first_generation) is list:
        evolution = first_generation[:]
    elif type(first_generation) is int:
        a = [[random.uniform(-100, 100) for _ in range(first_generation)]]
        evolution.extend(a)
    else:
        raise TypeError('Bad init data. Should be int or list')
    iters = 0
    stop=func(find_best(evolution,func))
    stop_kol=0
    it_shower = IterPrinter(find_best(evolution, func),func)
    n2 = len(first_generation)
    while True :
        iters += it_shower.next(1, find_best(evolution, func),func)
        evolution = disaster(crossover(mutation(evolution, func), func,queries,pairs,Q), func,
n2,pairs,D,queries,Q)
        stop1=func(find_best(evolution,func))
        if stop1!=stop:
            stop=stop1
            stop_kol=0
        else:
            stop_kol=stop_kol+1
        if is_ideal(func, evolution, eps)or stop_kol==10:
            ans = find_best(evolution, func)
            it_shower.stop()
            return ans, iters
```

Приложение 4. Функция жадного алгоритма

```
def first_solution(n1,D,queries,group,Q):
    print(group)
    n=len(group)
    res=[]
    res.append(0)
    P=[]
    for i in range(0,n):
        if group[i]<n1+1:
            P.append(group[i])
    Wv=[0]
    f=0
    k=0
    s=max(D[k])
    l=k
    w=0
    sum1=0
    while len(res)<2*n+1:
        if len (P)<1:
            break
        while f==0:
            P1=[]
            for h in P:
                P1.append(h)
            f1=0
            while f1==0:
                if len(P)==1:
                    k=P[0]
                    s=D[l][k]
                else:
                    for h in P1:
                        if D[l][h]<s:
                            s=D[l][h]
                            k=h
                if w+queries[k]<=Q:
                    f1=1
                    f=1
                    res.append(k)
                    sum1=sum1+s
                    l=k
                    s=max(D[k])
                    w=w+queries[k]
                    Wv.append(w)
                    if k<=n1:
```

```
                P.remove(k)
                P.append(k+n1)
            else:
                P.remove(k)
                s=max(D[l])
                k=l
        else:
            P1.remove(k)
            s=max(D[l])
    f=0
res.append(2*n1+1)
sum1=sum1+D[res[len(res)-1]][2*n1+1]
sum1=0
for jk in range(1,len(res)):
    sum1=sum1+D[res[jk-1]][res[jk]]
Wv.append(0)
```

Приложение 5. Функция динамической адаптации метода поиска с запретами

```
for rout in res_all:
    if len(rout)<din_k:
        res_all_new.append(rout)
        res_all_sum_new.append(res_all_distants[res_all.index(rout)])
        continue
    rout1=[]
    din_kj=int(len(rout)/din_k)
    din_kj_s=1
    for kj in range(0,din_k):
        rout1.append(rout[kj])
    res_f=rout1
    res_e=rout
    for jk in range(0,len(rout1)-1):
        res_e.remove(rout1[jk])
    group_remove=[]
    for t in res_f:
        group_remove.append(t)
    group_remove.remove(0)
    group=[]
    for t in res_e:
        group.append(t)
    if len(group)==0:
        resn1=[]
        sumn1=0
    else:
        group.remove(group[len(group)-1])
        if len(group)==0:
            resn1=[]
            sumn1=0
        else:
            group.remove(group[0])
            resn1,sumn1=tabu_search_new(res_e,group,group_remove,l,n1,D,queries,Q,rtabu)
    res11n=[]
    for t in range(0, len(rout1)):
        res11n.append(rout1[t])
    for t in range(1, len(resn1)):
        res11n.append(resn1[t])
    rout1n=[]
    for ig in range(0,2*n1+2):
        rout1n.append([0]*(2*n1+2))
    sum2=0
    for jf in range(1,len(res11n)):
```

```
        rout1n[res11n[jf-1]][res11n[jf]]=1
        sum2=sum2+D[res11n[jf-1]][res11n[jf]]
    din_kj_s=din_kj_s+1
    while din_kj_s<din_kj:
        if len(resn1)<k:
            break
        resn1.remove(resn1[0])
        if len(resn1)<k:
            break
        for f in range(0,k):
            rout1.append(resn1[f])
        for f in range(0,k-1):
            resn1.remove(resn1[0])
        group=[]
        for t in resn1:
            group.append(t)
        if len(group)==0:
            resn1=[]
            sumn1=0
        else:
            group.remove(group[0])
            if len(group)==0:
                resn1=[]
                sumn1=0
            else:
                group.remove(group[len(group)-1])
                group_remove=[]
                for t in rout1:
                    group_remove.append(t)
                group_remove.remove(0)
                resn1,sumn1=tabu_search_new(res_e,group,group_remove,l,n1,D,queries,Q,rtabu)
        din_kj_s=din_kj_s+1
```

Приложение 6. Таблица результатов работы алгоритмов

TS – метод поиска с запретами; DTS – динамически адаптированный метод поиска с запретами; GA – генетический алгоритм.

файл	Q	k	Лучшее решение			Среднее решение		
			TS	DTS	GA	TS	DTS	GA
1P1	30	3	1548,13	1544,57	1527,63	1554,27	1551,69	1534,79
	15	3	1639,2	1595,61	1527,63	1644,72	1601,13	1534,79
	10	3	1775,02	1756,29	1766,26	1775,02	1756,29	1771,84
	5	3	2257,1	2107,64	2115,05	2342,4	2175,43	2216,57
	30	5	1733,3	1711,31	1697,43	1853,45	1820,42	1750,51
	10	5	2005,74	1829,14	1821,61	2005,74	1834,49	1836,49
	5	5	2400,86	2249,55	2291,42	2427,71	2255,35	2307,06
	3	5	2927,51	2853	2832,18	2918,09	2863,75	2892,27
	30	10	2497,87	2493,72	2228,12	2531,13	2502,37	2309,64
	5	10	2659,84	2626,27	2686,63	2898,68	2700,29	2694,78
	3	10	3206,49	3206,49	3281,59	3369,9	3278,17	3315,2
11P1	30	3	1564,72	1482,49	1476,86	1573,82	1573,82	1485,44
	15	3	1573,82	1573,82	1478,77	1573,82	1573,82	1529,64
	10	3	1599,26	1573,82	1582,49	1599,26	1573,82	1597,06
	5	3	1707,82	1637,27	1639,34	1747,39	1637,27	1736,05
	30	5	1924,9	1924,9	1823,31	1935,28	1929,62	1827,49
	10	5	1924,9	1924,9	1823,31	1924,9	1924,9	1830,57
	5	5	2108,62	2014,79	2035,96	2173,15	2103,84	2125,44
	3	5	2348,43	2234,44	2236,44	2348,43	2234,44	2244,4
	30	10	2161,1	2154,73	2121,99	2426,6	2390,2	2183,11
	5	10	2273,92	2208,41	2193,29	2341,93	2251,97	2243,7
	3	10	2759,74	2756,08	2757,71	2901,64	2778,25	2779,15
21P1	30	3	870,46	865,86	869,09	881,51	872,69	870,87
	15	3	1071,52	1028,66	1103,65	1101,48	1053,24	1104,76
	10	3	1375,21	1279,32	1284,44	1393,41	1289,16	1292,46
	5	3	2189,66	1630,09	1657,99	2209,38	1676,58	1688,65
	30	5	1275,81	1275,81	1316,79	1275,91	1275,81	1319,25
	10	5	1440,54	1440,43	1404,13	1440,54	1440,43	1408,89

	5	5	2428,91	1593,39	1824,12	2430,01	1594,49	1828,86
	3	5	2844,22	2567,89	2620,05	2845,32	2568,99	2714,78
	30	10	2116,21	2116,16	1978,31	2116,21	2116,16	1982,15
	5	10	2422,77	2422,77	2450,33	2453,18	2434,87	2456,19
	3	10	3312,45	3201,07	3194,32	3349,81	3212,15	3305,31
31P1	30	3	887,13	882,84	886,43	1043,26	1038,52	896,15
	15	3	1008,21	1005,97	924,56	1048,36	1020,26	988
	10	3	1131,8	1130,13	1131,93	1199,22	1177,69	1171,57
	5	3	1494,57	1346,56	1393,61	1648,84	1462,9	1471,62
	30	5	1235,27	1108,65	1052,91	1278,03	1277	1056,19
	10	5	1329,79	1323,17	1320,47	1331,64	1330,6	1325,57
	5	5	1718,71	1631,58	1665,17	1783,36	1659,6	1678,93
	3	5	2093,51	1955,16	1977,24	2101,64	1974,31	1998,53
	30	10	1705,53	1703,29	1688,91	1923,83	1914,23	1885,06
	5	10	1915,7	1860,4	1862,09	2050,2	1986,31	1998,22
	3	10	2334,98	2185,92	2214,61	2334,98	2185,92	2217,31
41P1	30	3	750,61	747,78	730,09	767,84	755,39	751,23
	15	3	750,61	747,78	762,23	767,84	755,39	763,23
	10	3	763,74	760,92	822,58	933,62	929,5	887,45-
	5	3	1205,19	1105,24	1193,84	1223,24	1198,77	1201,4
	30	5	1083,06	1083,06	928,82	1083,06	1083,06	932,09
	10	5	1083,06	1083,06	929,03	1083,06	1083,06	934,09
	5	5	1451,22	1371,19	1455,11	1451,22	1371,19	1457,76
	3	5	1990,3	1867,19	1881,56	1993,68	1889,23	1898,09
	30	10	1518,44	1517,83	961,89	1518,44	1517,83	964,59
	5	10	1688,98	1604,64	1686,09	1696,5	1642,7	1687,09
	3	10	2263,22	2210,14	2266,38	2276,8	2275,67	2271,23
51P1	30	3	1162,78	1161,1	1154,27	1168,78	1168,1	1155,13
	15	3	1270,83	1263,19	1258,67	1276,79	1275,89	1276,55
	10	3	1331,23	1318,25	1317,87	1340,09	1329,86	1344,43
	5	3	1765,18	1691,38	1686,9	1770,26	1707,21	1698,66
	30	5	1322,42	1322,42	1221,84	1333,11	1330,83	1228,92
	10	5	1478,3	1334,28	1348,28	1484,42	1346,13	1365,81

	5	5	2013,2	1801,61	1867,36	2109,52	1854,35	1890,96
	3	5	2614,6	2400,42	2417,09	2856,57	2554,1	2484,39
	30	10	2205,22	2205,22	1948,45	2205,22	2205,22	1950,29
	5	10	2318,57	2306,76	2317,51	2336,35	2324,71	2351,95
	3	10	3324,09	3318,74	3315,88	3328,03	3321,92	3321,46
60P1	30	3	1270,24	1270,24	1182,46	1270,24	1270,24	1182,46
	15	3	1234,81	1234,81	1203,15	1267,19	1254,72	1263,42
	10	3	1234,81	1234,81	1203,15	1267,19	1254,72	1263,42
	5	3	1276,64	1199,47	1276,04	1291,23	1217,66	1302,31
	30	5	1456,75	1456,75	1371,51	1465,94	1458,14	1384,73
	10	5	1456,75	1456,75	1372,45	1465,94	1458,14	1488,33
	5	5	1521,38	1493,96	1487,86	1523,65	1504,94	1520,08
	30	10	2101,3	2097,28	1922,49	2164,44	2158,01	1957,43
	5	10	1903,4	1882,73	1875,29	1988,23	1982,07	1945,37
	3	10	2118,54	2107,98	2129,19	2153,09	2125,18	2131,65